UNIVERSITÉ DE MONTRÉAL

CENTRE DE RECHERCHE SUR LES TRANSPORTS

CASE POSTALE 6128, SUCCURSALE A, MONTREAL, QUEBEC, H3C 3J7

A GENERALIZED PERMANENT LABELLING ALGORITHM FOR THE SHORTEST PATH PROBLEM WITH TIME WINDOWS

by

Martin Desrochers and François Soumis*

* Département de mathématiques appliquées, École Polytechnique and Centre de recherche sur les transports, Université de Montréal

Cette a été publiée grâce à une subvention du fonds F.C.A.R. pour l'aide et le soutien à la recherche.

Université de Montréal
Centre de recherche sur les transports - Publication #394A
Mars 1985 - Révisé en août 1986

Dépôt légal - 4e trimestre 1986

ABSTRACT

The shortest path problem with time windows (SPPTW) consists of finding the least cost route between a source and a sink in a network $G=(N,A)$ while respecting specified time windows $[a_i, b_i]$ at each visited node. The duration d_{ij} of each arc is restricted to positive values while the cost c_{ij} of each arc $(i,j) \in A$ is unrestricted.

This article presents an efficient generalized permanent labelling algorithm to solve this problem. This new algorithm is based on the definition of the concept of a generalized bucket and on a specific order of handling the labels. The algorithm runs in pseudo-polynomial time. Problems with up to 2500 nodes and 250,000 arcs have been solved.

RÉSUMÉ

Le problème du plus court chemin avec contraintes d'horaires (PPCCCH) se pose lors de la fabrication d'horaires de véhicules devant respecter des fenêtres de temps. La solution étant composée du chemins disjoints visitant tous les clients, le PPCCCH permet de produire de tels chemins.

Cet article présente un algorithme de marquage permanent généralisé des étiquettes efficaces. Ce nouvel algorithme est basé sur la définition d'un concept de seau généralisé et d'un ordre de traitement des étiquettes et il fonctionne en un temps pseudo-polynomial. Des problèmes comportant jusqu'à 2500 noeuds et 250 000 arcs ont été résolus.

Unknown a few years ago, time windows constraints are rapidly becoming a popular way to model opening hours of customers, prefered delivery time, etc. in many scheduling and routing problems.

The problem first appeared as a sub-problem during the construction of school bus routes, where the number and composition of routes required to carry out all tasks must be found while minimizing total costs. Desrosiers, Soumis and Desrochers [5] use a column generation method to construct optimal routes covering the set of tasks to be carried out. At each iteration of the column generation method, the current solution is improved by inserting into the basis the least marginal cost route including a subset of tasks respecting the time window constraints. This route is obtained by solving the shortest path problem with time windows. (SPPTW).

Sorensen [14] proposes to solve the time constrained vehicle routing problem, i.e. a routing problem with capacity constraints on vehicles and time windows constraints on nodes, using the generalized assignment problem and the SPPTW as sub-problems.

1. THE PROBLEM

The shortest path problem with time windows consists of finding the least cost route between two nodes in a network while visiting each chosen node i within a specified time window $[a_i, b_i]$. Let G=(N,A) be a network where N is the set of nodes representing the tasks, a source p and a sink q. With each arc (i,j) is associated an unrestricted cost c_{ij} and a positive duration d_{ij}. Cost c_{ij} includes the inter-task cost from i to j and the cost of task i. Duration d_{ij} includes the inter-task duration from i to j and the duration of task i. Given the time windows $[a_i, b_i]$ for each

task, all arcs $(i,j) \in A$ are such that if task i is executed at the earliest possible time, task j may then begin before the end of its time window i.e. all arcs (i,j) respect the condition :

$$a_i + d_{ij} < b_j \qquad (1)$$

Even if the time windows constraints and the positive durations guarantee the finitness of feasible paths, it does not guarantee that feasible paths will be elementary. When generating columns, non-elementary paths, i.e. paths where at least a node is visited twice, are often encountered in early solutions when node's marginal values are unbalanced, creating temporary negative cost cycles that will disappear as the marginal values settle to their final values. The final solution is very unlikely to be a non-elementary path. Thus the presence of negative cycles can be neglected in this context.

The SPPTW is NP-Complete (from shortest weight-constrained path Garey and Johnson [6]).

2. LITERATURE REVIEW

The unconstrained shortest path problem has a considerable importance in transportation models and is the subject of an enormous number of papers, see Gallo and Pallottino for a survey. The solution to the shortest path problem is a directed spanning tree T of G=(N,A) rooted at source p and let label C_i be the cost of the unique path in T from p to i, $i \in N$. T is a shortest path tree with origin p if and only if the Bellman's equations hold :

$$C_i + c_{ij} - C_j \geq 0 \qquad \text{for all } (i,j) \in A . \qquad (2)$$

Gallo and Pallottino note that all the shortest path algorithms use dynamic programming and perform the same operations :

1. Initialize a directed tree T rooted at p and let C_i be the cost of the path from p to i in T for all $i \in N$;

2. Let $(i,j) \in A$ be an arc for which condition (2) is not satisfied, then adjust label $C_j := C_i + c_{ij}$ and update the tree T replacing the current arc incident into node j by arc (i,j);
3. Repeat step 2 until conditions (2) are sastisfied for all arcs.

The main aspect in the implementation of this procedure is how to select an arc at step 2 to verify condition (2). In most algorithms, a node k is selected and treated i.e. for all arcs $(k,j) \in A$ step 2 is performed. Let Q be the set of candidate nodes, there are many selection rules for node $k \in Q$:

1. FIFO (First-In-First-Out) : the oldest element in Q is selected and treated. The set Q is represented by a queue. New nodes are inserted at the tail of the queue and the node at the head of the queue is selected and treated;
2. LIFO (Last-In-Last-Out) : the newest element in Q is selected and treated. The set Q is represented by a stack. New nodes are inserted at the top of the stack and the node at the top is selected and treated;
3. Best-First : the node $k \in Q$ with the smallest label C_k is selected and treated.

Gallo and Pallottino [6] report that in practice two derived rules perform very well. In the L-deque method, the set Q is represented by a deque (double-ended queue). The first time a node is inserted in Q, it is added at the tail of the deque. Subsequent insertions of this node occur at the head of the deque. The node at the head of the deque is selected and treated. The worst case complexity of L-deque is exponential. The L-2queue method is similar but has a polynomial complexity of order $O(|N|^2|A|)$. It uses a double queue to represent the set Q. The first time a node is inserted in Q, it is added at the tail of the second queue. subsequent insertions of this node occur at the tail of the first queue. The node at the head of the first queue is selected and treated.

The FIFO rule, LIFO rule and derived rules are often referred to as defining label-correcting methods. The Best-First rule is used to define label-setting methods.

2.1 The shortest path problem with additional constraints

This is the special case most frequently discussed [10,12,8,1,9]. This is the case with all a_i equal to zero and all b_i equal. Two families of algorithms have been proposed to solve this problem, one using dynamic programming and the other involving Lagrangian relaxation.

The Lagrangian relaxation method proposed by Minoux [12] involves the solution of a shortest path problem with costs modified by the addition of a multiplier associated with the supplementary constraints. This produces a feasible solution and a lower bound on the value of the optimal solution. Handler and Zang [9] propose a similar method and add a second step which generates an optimal solution. This second step uses the calculation of the K^{th} shortest path with modified costs to close the duality gap. This method can solve low density problems (1% to 10%) with 50 to 500 nodes. Beasley and Christofides [2] present a similar method for the multiple resources case. They explore the duality gap with a branch and bound method. They report solving low density problems (2% to 10%) with 100 to 500 nodes and up to 10 resources.

Ribeiro discusses another extension [13] : the shortest path problem with additional double inequalities. Ribeiro proposes several dual methods for solving this problem. These methods are based on Lagrangian relaxation and the subgradient method for the adjustment of the Lagrange multipliers. They give feasible solutions and optimality can be proved for problems of average to high density (25% to 100%) having 50 to 100 nodes.

The dynamic programming approach was first proposed by Joksch [10] in 1966. His method solves the recurrence equations of efficient solutions using a generalization of the label-correcting method (FIFO rule) for the shortest path problem. Minoux [12] proposes a method based on a generalization of the label-setting method possible because of the non-negativity of the costs and durations. Aneja, Aggarwal and Nair [1] propose a similar

method for solving a version of the problem with several supplementary constraints. This method involves the computation of a relevant subset of efficient solutions at each node. The method uses several bounds to limit the number of solutions examined. Jaffe [9] proposes a method based on the label-correcting method to calculate the solution at all possible states, rather than only considering efficient solutions. He makes a detailed study of the effect on solution accuracy of only calculating some of the possible states.

2.2 The shortest path problem with time windows

The SPPTW was formulated by Desrosiers, Pelletier and Soumis [4] as a subproblem of a route construction problem [5]. The authors prove that at least one of the optimal solutions of the SPPTW is integer, and they formulate the following optimality principle : for a given path X_{pj} from source p to node j, if this path is efficient and if arc (i,j) is the last arc of X_{pj}, then the sub-path X_{pi} is an efficient path as defined in section 3. This optimality principle is used in an algorithm called SPTW based on dynamic programming. This a generalization of a label-correcting method called L-deque by Gallo and Pallottino [6]. This method generally allows the efficient treatment of the nodes, However in the worst case, its complexity is exponential. This method is sensitive to the initial ordering of the data, and the algorithm can be accelerated by sorting the data in increasing order of starting times of the time windows. This algorithm can solve problems of low to medium density (15%- 30%) with up to 300 nodes.

Almost all the previous algorithms for the shortest path problem with additional constraints [1,8,9,10,12,13] deal with positive costs and duration. Only Beasley and Christofides [2] do not use this assumption, they only need that there is no negative cost circuit. The SPPTW has no such restrictions on cost because the time windows forbid infinite cycling.

3. THE GENERALIZED PERMANENT LABELLING ALGORITHM

The SPTW algorithms of Desrosiers et al. [4] could be improved by using a different rule for node selection. Instead of choosing a generalization of the L-deque method, we could obtain the first pseudo-polynomial algorithm for the SPPTW by generalizing the L-2queue method [6]. The resulting algorithm would run in $O(D^3)$ operations where $D = \sum_{i=1}^{n} (b_i - a_i + 1)$ is the maximum number of efficient paths.

In this section, we present a further improvement to the SPTW algorithm resulting in a label-setting method running in $O(D^2)$ operations. We gain an order of magnitude by defining a Best-First rule of treatment of the efficients labels and by using generalized buckets.

3.1. Paths and label treatment

With each path X_{pj} from the origin p to the node j satisfying time windows, is associated a (time, cost) label corresponding respectively to the arrival time at node j and the cost of the path X_{pj}. These labels will be denoted by (T_i^k, C_i^k) to indicate the characteristics of the k^{th} path from p to i, where the indices k and i may be dropped when the context is unambiguous.

These labels are calculated iteratively along the path $X_{pj} = (i(0), i(1), i(2), \ldots, i(L))$, where $i(0) = p$ and $i(L) = j$, as follows :

$$T_{i(0)} = 0$$

$$C_{i(0)} = 0$$

$$T_{i(\ell)} = \max\{a_{i(\ell)}, T_{i(\ell-1)} + d_{i(\ell-1)i(\ell)}\} \qquad \ell = 1, \ldots, L$$

$$C_{i(\ell)} = C_{i(\ell-1)} + c_{i(\ell-1)i(\ell)} \qquad \ell = 1, \ldots, L$$

Definition 1. Let X^1 and X^2 be two different paths from p to j with associated labels (T^1, C^1) and (T^2, C^2). Then X^1 dominates X^2 or $(T^1, C^1) < (T^2, C^2)$ if and only if $(T^2, C^2) - (T^1, C^1) \geq (0, 0)$ and $(T^1, C^1) \neq (T^2, C^2)$.

This dominance relation is not a total ordering and does not allow all paths to be ordered. But, it does allow us to conclude that the undominated efficient path X_{pj} is the shortest path arriving at node j at time T_j or before. This implies the possibility of several efficient paths for each node. Figure 1 illustrates the relation between several different (time, cost) labels associated with efficient paths (X^1 to X^4) and others which are dominated (X^5 to X^8). EFF(Q) denote the set of efficient labels among Q.

This relation allows the definition of the cost of the feasible path as a function of arrival time. Figure 1 shows this cost-time function for a given node.

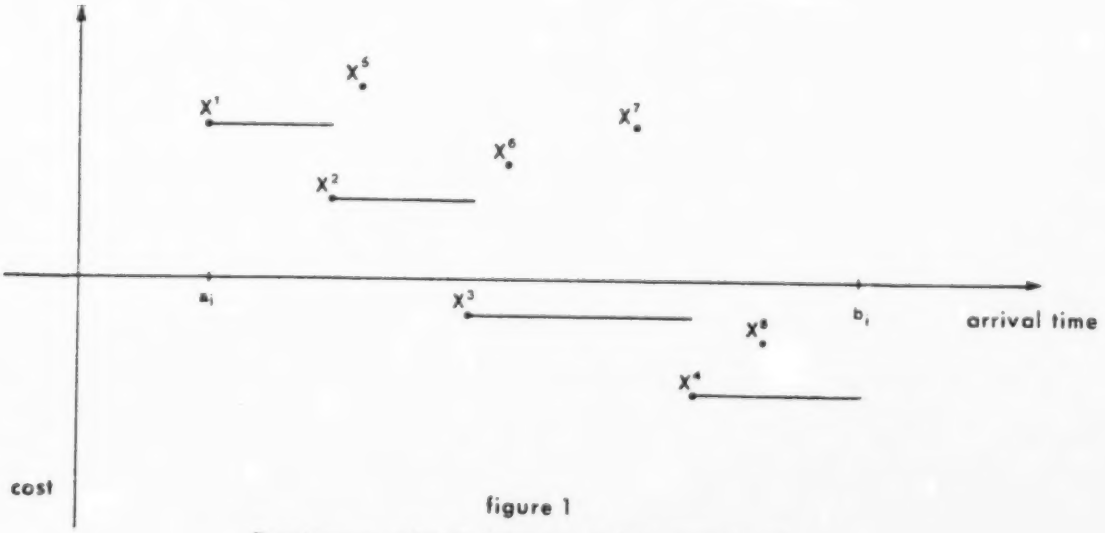

figure 1
Dominance relation between labels associated with different paths

Let (T_i^k, C_i^k) be an efficient label at node i and its associated path X_{pi}. When arc $(i,j) \in A$ is added to the path X_{pi}, a new path X_{pj} for node j is obtained if $T_i^k + d_{ij} \leq b_j$. The label associated to this path is :

$$(Tj, Cj) = (\max[aj, T\overset{k}{i}+dij], C\overset{k}{i}+cij)$$

The label is added to the set Q_j and this set is updated :

$$Q_j := \mathrm{EFF}(Q_j \cup \{T_j, C_j)\}).$$

This treatment is done for all arcs $(i,j) \in A$ incident to node i.

3.2 A new order of treatment of the labels

In unconstrained shortest path problems, the notions of "node" and "label" are closely linked. The order of treatment is thus defined simultaneously for nodes and labels. In the case of the constrained shortest path problem, the two notions are distinct, as a set of labels is associated with each node. In general, when algorithms are developed for these problems, the treatment order focusses on the nodes and the resulting algorithm is a label-correcting method [10,9,4]; for the special case where costs and durations are positive, the resulting algorithm is a label-setting method [12,1].

The existence of a label-setting for the SPPTW depends on the existence of an order of treating the labels with a qualification condition on the arcs such that it is impossible to improve a label which has been previously treated.

The well-known lexicographic ordering is a total ordering in R^2 which is compatible with the partial dominance ordering, i.e. the following three propositions hold :

1) for all $a, b \in R^2$, if $a < b$, then $a \overset{L}{\leq} b$.
2) for all $a, b \in R^2$, if $a > b$, then $a \overset{L}{\geq} b$.
3) for all $a, b \in R^2$, if $a \overset{L}{<} b$, then $a > b$.

Theorem 1

Let P be the set of labels already treated and
let T be the set of untreated labels.
If 1) all arcs $(i,j) \in A$ are lexicographically positive;
2) the labels of T are treated in increasing lexicographic order

then

- for all $a \in P$, and all $b \in T$, $a \overset{L}{<} b$,
- the treatment of the smallest label of T cannot improve a label in P.

Proof : The successors of any given label being treated are all lexicographically greater than this label. Using induction on the cardinality of P and that any element of T is a successor of an element of P, we derive the conclusions easily. □

3.3 The concept of a generalized bucket

The criterion for choosing the next label to be treated can be improved by using a generalization of the concept of a "bucket" introduced by Denardo and Fox [3]. A bucket is a list of nodes for which the value of the label lies within an interval. In the simple case, the p^{th} bucket is made up of all nodes whose label values are included in the semi-open interval $[pm,(p+1)m)$ where m is the width of the bucket which is fixed at

$$m = \min_{(i,j) \in A} \{c_{ij}\} .$$

The search for the smallest temporary label is thus replaced by the search for an element of the first bucket to contain the temporary labels.

We use the same concept of a bucket by defining the width of the bucket in $\mathbb{R}^2$ as

$$(m_d, m_c) = \operatorname*{minlex}_{(i,j) \in A} \{(d_{ij}, c_{ij})\}$$

Let F(T) be the next label to be handled according to step 2 and let B(T) be the generalized bucket defined by :

$$F(T) = \operatorname*{minlex}_{(T_i^k, C_i^k) \in T} \{(T_i^k, C_i^k)\}$$

$$B(T) = \{(T_i^k, C_i^k) \in T \mid F(T) \overset{L}{<} (T_i^k, C_i^k) \overset{L}{<} F(T) + (m_d, m_c)\}.$$

We need to show that replacing the treatment of F(T) by the treatment of an element of B(T) does not alter the optimality of the algorithm.

Theorem 2. If $(m_d, m_c) \overset{L}{>} (0,0)$, then the elements of the generalized bucket B(T) cannot be improved during the treatment of an element of B(T).

Proof. Suppose it were possible to improve an element (T_i^k, C_i^k) of B(T) by treating another element (T_i^ℓ, C_i^ℓ) of B(T)

$$(T_i^\ell, C_i^\ell) + (d_{ij}, c_{ij}) \prec (T_j^k, C_j^k)$$

$$\Rightarrow (T_i^\ell, C_i^\ell) + (d_{ij}, c_{ij}) \overset{L}{<} (T_j^k, C_j^k).$$

On the other hand, $(T_j^k, C_j^k) \overset{L}{<} F(T) + (m_d, m_c)$ as $(T_j^k, C_j^k) \in B(T)$

$$(T_i^\ell, C_i^\ell) + (d_{ij}, c_{ij}) \overset{L}{<} F(T) + (m_d, m_c)$$

$$(T_i^\ell, C_i^\ell) \overset{L}{<} F(T) + (m_d, m_c) - (d_{ij} c_{ij})$$

$$(T_i^\ell, C_i^\ell) \overset{L}{<} F(T) \text{ as } (m_d, m_c) \overset{L}{<} (d_{ij}, c_{ij})$$

which contradicts the definition of B(T). It is therefore impossible to improve the elements of B(T). □

3.4 The generalized label-setting algoritm

To use the results of the two previous sub-sections in the development of an algorithm, it is sufficient to verify that for the SPPTW, the costs on the arcs are lexicographically positive. The pairs (duration, cost) were placed in this order because in the problem with time window constraints, the durations d_{ij} (duration of task i plus the inter-task duration) are always positive. This is sufficient to satisfy the qualification condition on the arcs without any restrictions on the costs c_{ij}. The algorithm can be described as follows :

Step 1 : Initialization

$$Q_i = \begin{cases} \{(0,0)\} & i=p \\ \emptyset & \text{for all } i \in N,\ i \neq p \end{cases}$$

$$P_i = \emptyset \quad \text{for all } i \in N$$

Q_i is the set of permanent labels for node i,

P_i is the set of candidate labels for node i.

Step 2. Find the current bucket

Find F(T) the label $\{(T_i^k, C_i^k)\}$ of lexicographically minimum cost from the set $T = \bigcup_{i \in N} (Q_i - P_i)$. If $T = \emptyset$, stop.

. Calculate the upper bound of B(T).

Step 3. Find the next label to be treated

. Find one element of B(T).

. If B(T) is empty, go to step 2.

Step 4. Treatment of label (T_i^k, C_i^k)

For all successors j of node i do

begin

if $T_i^k + d_{ij} < b_j$ (time windows satisfied)

then $(T_j, C_j) = (\max(a_j, T_i^k + d_{ij}),\ C_i^k + c_{ij})$

$$Q_j = \text{EFF}(Q_j \cup \{(T_j, C_j)\})$$

end.

$$P_i = P_i \cup \{(T_i^k, C_i^k)\}$$

go to step 3.

In the case of the SPPTW, as the durations d_{ij} are strictly positive, the order of treatment depends on the value of tmin where

$$tmin = \min_{(i,j)\in A} d_{ij}.$$

Figure 2 illustrates the time window associated with a given node. The nodes are sorted in increasing order of the starting times of their time windows. Although this is not essential for the implementation of the algorithm, it can improve the behaviour of the algorithm on large problems as we shall see in the section on numerical results.

The example shown includes five buckets. The time windows and the labels associated with each node have been divided into subsets according to the boundaries of the buckets. This generates 24 subsets Q_i which may each contain labels. The order of treatment of the labels is determined by their subset membership. The treatment of the first bucket begins by the treatment of all labels in Q_1. It continues with the treatment of the labels Q_2 and so on until all the labels of Q_4 have been handled. The second bucket is then treated starting with Q_5. The algorithm terminates after the treatment of Q_{24}.

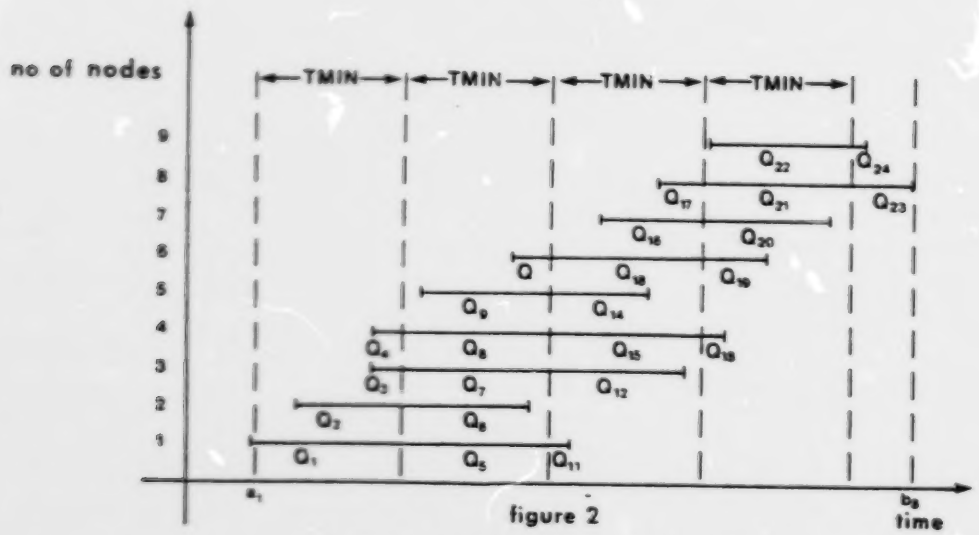

figure 2

Order of treatment of labels according to their position in the time window

4. THE COMPLEXITY OF THE ALGORITHM

The complexity of the algorithm presented depends on the number of possible labels, because potentially, each label will be treated once and only once. Complexity also depends on the data structure used to implement the sets Q_i. We examine here three different data structures, each having desirable properties under different circumstances.

Let $D = \sum_{i \in N} (b_i - a_i + 1)$: the number of possible labels,

$d = \max_{i \in N} (b_i - a_i + 1)$: the widest time window,

$n = |N|$: the number of nodes,

$m = |A|$: the number of arcs.

In the first data structure, a linear list, sorted in lexicographic order, is used to represent the sets Q_i. The memory requirement is $O(m+D)$ in the worst case. However, these lists are efficient only if a small proportion of labels actually exists, allowing reductions in the memory required and the work to be carried out.

Step 1 is executed once and is of order n. Step 2 is executed at most D times (in the case where each bucket contains only one label) and each execution requires at most $O(n)$ operations. In the same worst case, step 3 is executed at most D times and retrieval of one element of B(T) requires a verification of the beginning of each list of labels ($O(n)$ operations). The treatment of the chosen label (step 4) is carried out at most D times and the number of successors for each label treated is at most n. For each of these successors, the merger and reduction of the set of undominated labels associated with node i, requires the comparison of the new label with at most $b_i - a_i + 1$ other labels. Overall, for all successors, this requires $O(D)$ operations. The complexity of this implementation is therefore $O(D^2)$. However, the work to be carried out is pro-

portional to the square of the number of labels actually present in the solution. This approach is satisfactory if this number is small relative to D.

The two other approaches use a table to represent the sets Q_i. The table includes the minimum cost of the paths for each node and each possible arrival time at this node. This table includes all efficient paths as well as some dominated paths. To carry out the reduction, it is sufficient to carry out a dominance test at each node at the end of the algorithm. This test has a complexity of O(d). This allows us at step 4 to check only whether the cost of the new path is less than that of the old path. These two approaches use at least O(m+D) words of memory. They are efficient when the number of labels is large relative to D.

For dense graphs, the use of a table allows the application of an approach similar to the first one described. The use of a table produces after the modification of step 4, a complexity of O(nD) for the algorithm.

The third approach which is efficient for low density graphs uses a heap in addition to the table. The heap provides access in constant time to the lexicographically minimum label within the table. The use of the heap requires a modification of the algorithm. At step 2, during the search for the next label to be treated, we obtain in constant time the lexicographically minimum label. At step 4, the treatment of labels is modified as we must, in addition, check its dominance in the table, and reorder the heap which requires O(log md) operations. As step 4 is executed at most nd times and as it requires the investigation of at most md successors, the global complexity of this step and of the algorithm is O(md log nd) operations.

These three results for problems with lexicographically positive arcs can be compared with that of Jaffe [9] for the problem with positive

costs and durations. This algorithm uses tables of O(nD) words of memory and has a complexity of $O(n^3d)$.

Table 1 : Summary of the complexity results

implementation	Memory complexity	Time complexity
with linear list	O(m+D)	$O(D^2)$
with a table	Ω(m+D)	O(nD)
with a table and heap	Ω(m+D)	O(md log nd)
Jaffe	Ω(nD)	$O(n^3d)$

5. NUMERICAL RESULTS

The permanent generalized labelling algorithm (GPLA) presented in section 3 was implemented in FORTRAN 77, according to the first approach, and tested on a CDC CYBER 173 to compare its performance with that of the SPTW algorithm. The comparison between these two algorithms was carried out by varying four parameters in the test problems :

1. The number of nodes n making up the network. We solve problems with 100, 250, 500, 1000 and 2500 nodes.
2. The average number of arcs per node. We choose randomly from among the feasible arcs according to (1) so as to obtain a total of around 10n, 25n, 50n or 100 n arcs in the network.
3. The width of the time windows. The time window at each node has a width of 10, 25, 50 or 100 time units with the beginnings a_i of the tasks being distributed uniformly between 0 and 100.
4. The fraction of nodes accessible from the fictional source. The 0.1*n, 0.25*n, 0.5*n on n nodes which may begin a path are those without predecessor or for which the a_i are minimal.

The first three parameters mainly determine the characteristics of the problems to be solved (number of nodes $|N|$, number of arcs $|A|$ and the

number of efficient solutions ($|E_s|$) and the fourth parameter mainly influences the initialization of the algorithm.

Three different test problems were generated for each feasible combination of the first three parameters. These problems were produced using the method proposed by Desrosiers, Pelletier and Soumis [4].

The problems were produced using the following parameters :

1. the tasks are dispersed in the square [0,70] x [0,70] according to a uniform distribution;
2. the durations d_i of tasks are distributed uniformly over the interval [5,15];
3. the inter-task time t_{ij} is set equal to the Euclidean distance between the tasks i and j plus the duration of task i;
4. the inter-task cost c_{ij} is equal to t_{ij} less a large constant (33,333). This constant represents the marginal cost of a task during column generation. The chosen constant is such that most three task path have a negative cost.

The computer used did not allow all nodes, arcs and labels to be stored in central memory. The nodes are stored in central memory, while the arcs are divided into pages and only one page is stored in central memory. When the execution of the algorithm requires the arcs from another page, this new page is entered into central memory. This process is similar to the creation of virtual memory which is available on other computers. The time required for memory management is calculated separately from algorithm execution time. Table 2 shows the comparison of the memory management time for the two algorithms for problems occupying two, three and five pages (according to the number of arcs) and having 500 nodes. Note that memory management time is generally much greater for SPTW than for GPLA. Certain problems could not be solved by SPTW because of frequent page default.

TABLE 2 - COMPARISON OF MEMORY TIME

			SPTW				GPLA		
D	|A|	$\|N_d\|$	# OF PROBLEMS SOLVED	# OF REIN-SERTIONS	MEMORY TIME (SEC)	EXECUTION TIME (SEC)	# OF PROBLEMS SOLVED	MEMORY TIME (SEC)	EXCUTION TIME (SEC)
5500	12487	50	3	1525	34.30	1.15	3	.43	.99
5500	12487	125	2	838	16.78	1.28	3	.31	1.26
5500	12487	250	3	200	.24	1.37	3	.29	1.31
5500	12487	500	3	3	.32	1.34	3	.29	1.34
5500	24982	50	3	2591	46.41	2.86	3	.99	1.83
5500	24982	125	3	1046	16.88	2.65	3	.57	2.60
5500	24982	250	3	261	.49	2.76	3	.57	2.62
5500	24982	500	3	6	.65	2.49	3	.60	2.59
5500	49984	50	3	3947	43.67	8.99	3	1.89	3.86
5500	49984	125	3	1701	15.75	6.08	3	1.29	5.13
5500	49984	250	3	240	.59	5.58	3	1.22	5.30
5500	49984	500	3	10	.99	5.39	3	1.23	5.27
13000	12527	50	3	1989	45.78	2.21	3	.83	1.53
13000	12527	125	3	1051	29.51	1.77	3	.47	2.21
13000	12527	250	3	438	2.49	2.38	3	.46	2.32
13000	12527	500	3	151	2.14	2.18	3	.49	2.28
13000	24907	50	3	3476	62.65	6.92	3	1.91	3.50
13000	24907	125	3	2088	41.73	4.85	3	1.28	4.84
13000	24907	250	3	729	5.86	5.52	3	1.23	4.96
13000	24907	500	3	244	6.79	4.71	3	1.13	5.05
13000	49988	50	3	4160	88.29	18.41	3	3.42	8.16
13000	49988	125	3	1905	47.78	11.49	3	2.33	10.49
13000	49988	250	3	707	12.11	12.33	3	2.17	10.90
13000	49988	500	3	355	18.36	10.02	3	2.30	10.73
25500	12501	50	3	3374	125.89	5.19	3	1.52	3.06
25500	12501	125	3	1981	86.73	4.46	3	1.61	4.06
25500	12501	250	3	1096	23.90	5.54	3	1.69	4.18
25500	12501	500	3	741	14.55	5.15	3	1.55	4.29
25500	24975	50	2	4702	240.27	9.75	3	3.26	7.69
25500	24975	125	3	2862	105.05	15.27	3	3.23	9.65
25500	24975	250	3	1388	48.76	12.59	3	3.49	9.47
25500	24975	500	3	1064	43.50	11.87	3	3.40	9.66
25500	49953	50	0	0	.00	.00	3	6.73	17.95
25500	49953	125	3	3916	159.22	40.04	3	6.22	21.35
25500	49953	250	3	1937	86.53	31.00	3	6.35	21.27
25500	49953	500	3	1339	93.35	26.18	3	6.71	21.04
50500	12477	50	1	4511	152.11	18.31	3	2.74	6.99
50500	12477	125	2	3623	106.94	19.27	3	1.93	8.80
50500	12477	250	2	2553	79.60	15.62	3	1.65	9.85
50500	12477	500	3	2477	38.75	17.86	3	1.72	9.91
50500	25074	50	0	0	.00	.00	3	6.12	18.58
50500	25074	125	2	4996	306.25	44.77	3	5.07	21.53
50500	25074	250	2	4483	199.50	50.34	3	4.71	22.67
50500	25074	500	3	3710	159.89	43.97	3	4.53	22.91
50500	49913	50	0	0	.00	.00	3	11.70	49.79
50500	49913	125	0	0	.00	.00	3	9.14	57.62
50500	49913	250	1	4575	218.48	165.76	3	8.29	58.16
50500	49913	500	3	4267	323.86	120.73	3	7.71	58.64

Note that the memory management time of SPTW depends on the number of nodes $|N_d|$ that are accessible from the source. For initialization of SPTW, the list of nodes to be treated is the list of nodes of $|N_d|$ in increasing order of the starting times of the time windows. As this is also the order of classifying the nodes on the pages, this means that the longer the initial list of nodes to be treated, the more the order of first treatment of the nodes corresponds to the order of the nodes on the pages and the fewer reinsertions have to be made with fewer page defaults. The treatment order defined in Section 5 means that GPLA encounters few new page references.

Table 3 contains various statistics on the execution of the two algorithms. Algorithm GPLA solved all problems except those whose solution involved too many efficient labels - thus exceeding available memory capacity. Algorithm SPTW was unable to solve certain problems solved by GPLA because of too many new page references; these problems included 1% of those with 250 nodes, 13% of those with 500 nodes and 30% of those with 1000 nodes. In the expectation that SPTW would be able to solve less than 25% of the 2500 node problems, these problems were not tested with SPTW. The unsolved problems being the most difficult ones, comparisons of execution time sometimes is difficult.

The three variables shown in Table 3 are the number of temporary labels produced $|E_T|$, memory management time, and algorithm execution time. By only handling successors of efficient solutions, GPLA guarantees that the number of temporary labels is minimized. Because of possible reinsertions, SPTW may produce more temporary labels. In fact, in several cases, we observe an increase of 50% over the number of temporary labels created by GPLA, with an increase of over 100% in a few cases.

TABLE 3 - EXECUTION STATISTICS FOR THE TWO ALGORITHMS (Part I)

				SPTH							GPLA					
							Memory management time		Execution time				Memory MGMT time		Execution time	
|N|	|D|	|A|	$\|E_s\|$	# of problems solved	$\|E_T\|$	# of rein-sertions	MEAN (sec)	MAX (sec)	MEAN (sec)	MAX (sec)	# of problems solved	$\|E_T\|$	MEAN (sec)	MAX (sec)	MEAN (sec)	MAX (sec)
100	1100	1004	149	12	1282	46	-	-	.11	.13	12	1229	-	-	.06	.11
100	1100	2171	184	12	3177	80	-	-	.24	.26	12	3023	-	-	.16	.21
100	2600	963	271	12	1716	93	-	-	.16	.21	12	1599	-	-	.12	.18
100	2600	2520	349	12	6077	129	-	-	.47	.58	12	5421	-	-	.35	.41
100	5100	973	456	12	8842	142	-	-	.66	.86	12	7761	-	-	.45	.56
100	5100	2482	669	12	2870	180	-	-	.30	.37	12	2433	-	-	.23	.28
100	5100	4988	917	12	11650	287	-	-	1.16	1.33	12	9029	-	-	.70	.83
100	5100	9285	896	12	34750	521	-	-	3.33	4.51	12	24813	-	-	1.86	2.17
100	10100	991	1001	12	36034	264	-	-	3.30	3.73	12	26003	-	-	1.94	2.33
100	10100	2524	1300	12	7348	412	-	-	1.00	1.13	12	5118	-	-	.55	.66
100	10100	5035	1713	12	33307	737	-	-	4.59	6.61	12	18248	-	-	1.65	1.98
100	10100	8725	2032	12	117341	1118	-	-	16.23	24.07	12	48732	-	-	4.33	5.71
				12	226901	775	-	-	26.29	32.63	12	101954	-	-	8.94	11.25
250	2750	2521	399	12	3356	153	-	-	.30	.34	12	3215	-	-	.27	.36
250	2750	6249	492	12	10156	322	-	-	.74	1.00	12	9090	-	-	.64	.79
250	2750	12478	578	12	24611	532	7.67	24.52	1.31	1.45	12	20642	.35	.50	1.17	1.37
250	2750	13636	590	12	27424	549	5.43	17.38	1.49	1.77	12	22947	.32	.53	1.32	1.47
250	6500	2510	657	12	4403	231	-	-	.42	.50	12	4139	-	-	.39	.49
250	6500	6220	955	12	17000	501	-	-	1.39	1.88	12	14595	-	-	1.08	1.23
250	6500	12460	1193	12	45080	831	17.17	48.75	2.67	3.08	12	36250	.65	1.09	2.25	2.67
250	6500	20312	1322	12	83844	699	5.30	14.48	5.63	8.30	12	68097	.65	1.06	4.11	4.91
250	12750	2537	1170	12	7344	397	-	-	.80	.96	12	6399	-	-	.68	.81
250	12750	6200	1752	12	30115	839	-	-	3.29	4.22	12	23288	-	-	2.00	2.41
250	12750	12561	2178	12	87234	1084	43.39	107.66	6.68	7.85	12	61079	1.04	1.44	4.63	5.30
250	12750	25018	2691	12	245666	1603	62.38	170.67	20.14	40.97	12	151775	2.47	3.66	10.91	12.72
250	25250	2430	2148	12	16220	996	-	-	2.24	2.55	12	11541	-	-	1.31	1.55
250	25250	6241	3535	12	88171	2209	-	-	13.66	18.67	12	48878	-	-	4.69	5.41
250	25250	12585	4225	12	292451	3751	132.23	300.74	39.15	73.66	12	122300	2.36	3.07	10.61	14.32
250	25250	25092	5231	10	627821	3150	152.61	306.18	78.13	139.06	12	287493	5.69	7.21	29.44	34.05

TABLE 3 - EXECUTION STATISTICS FOR THE TWO ALGORITHMS (Part II)

				SPTH						GPLA					
							Memory management time		Execution time				Memory MGMT time		Execution time	
|N|	|D|	|A|	$\|E_s\|$	# of problems solved	$\|E_T\|$	# of rein-sertions	MEAN (sec)	MAX (sec)	MEAN (sec)	MAX (sec)	# of problems solved	$\|E_T\|$	MEAN (sec)	MAX (sec)	MEAN (sec)	MAX (sec)
500	5500	5044	793	12	6820	361	-	-	.60	.68	12	6421	-	-	.55	.62
500	5500	12487	959	11	20745	624	12.56	39.79	1.28	1.42	12	18565	.33	.47	1.22	1.41
500	5500	24982	1071	12	49423	976	16.11	54.34	2.69	3.56	12	41237	.68	1.13	2.41	2.76
500	5500	49984	1181	12	117122	1475	15.25	59.39	6.51	9.63	12	91331	1.41	1.97	4.89	5.60
500	13000	4995	1333	12	9371	512	-	-	.90	.99	12	8632	-	-	.80	.93
500	13000	12527	1757	12	32955	907	19.98	47.13	2.14	2.43	12	28464	.56	.93	2.08	2.39
500	13000	24907	2141	12	88074	1634	29.26	66.20	5.50	7.23	12	71149	1.39	2.22	4.59	5.25
500	13000	49988	2488	12	216772	1782	41.64	108.07	13.06	21.71	12	167179	2.56	3.57	10.07	11.21
500	25500	5033	2391	12	15161	804	-	-	1.73	1.95	12	13057	-	-	1.39	1.63
500	25500	12501	3521	12	63737	1798	62.77	142.76	5.08	6.03	12	47805	1.59	1.94	3.90	4.43
500	25500	24975	4292	11	175573	2304	97.50	265.40	12.61	17.48	12	120484	3.34	3.91	9.12	10.72
500	25500	49953	4939	9	433938	2397	113.03	206.95	32.41	48.69	12	284001	6.50	7.45	20.40	22.81
500	50500	4952	4654	9	35828	1974	-	-	5.11	5.93	12	24598	-	-	2.84	3.59
500	50500	12477	6600	8	154936	3037	80.18	152.11	17.71	21.32	12	93566	2.01	3.06	8.89	10.66
500	50500	25066	8207	7	468973	4298	213.02	395.16	46.02	61.45	12	235377	5.11	6.60	21.42	24.57
500	50500	49913	10682	4	1271690	4344	297.52	357.68	131.98	165.76	12	611655	9.21	13.23	56.05	62.81
1000	11000	9993	1580	12	13282	659	18.91	135.30	1.16	1.32	12	12618	.14	.44	1.09	1.21
1000	11000	25015	1868	12	41005	1378	23.46	77.33	2.61	2.91	12	36063	.77	1.01	2.49	2.79
1000	11000	50129	2106	7	93420	1303	59.92	242.48	5.59	7.10	12	81026	1.90	2.59	4.98	5.67
1000	11000	99739	2438	8	220494	1855	39.00	196.74	11.91	14.06	12	187361	3.74	5.17	10.71	12.23
1000	26000	10018	2637	7	18431	760	8.42	36.97	1.68	1.92	12	16774	.42	.77	1.62	1.88
1000	26000	25058	3745	12	69955	2014	45.17	146.62	4.43	5.30	12	60312	1.17	1.78	4.73	5.41
1000	26000	50147	4417	9	173245	2562	76.92	190.97	10.04	13.81	12	142898	2.61	3.93	10.23	11.13
1000	26000	100103	5225	9	446126	3453	115.12	300.26	26.60	43.25	12	347727	6.19	9.20	23.03	24.89
1000	51000	10017	4723	8	30815	1406	8.62	42.06	3.24	3.67	12	25856	.58	1.16	2.93	3.34
1000	51000	25080	6895	12	127529	3607	171.00	377.51	8.46	14.39	12	95857	2.47	3.58	8.82	9.80
1000	51000	50052	8607	7	331961	3033	166.82	191.25	19.73	30.76	12	240228	6.95	9.86	19.94	22.53
1000	51000	99959	10720	4	856061	3237	289.56	386.90	49.97	63.86	12	612340	16.66	17.32	47.62	53.62
1000	101000	10012	9224	7	71637	3879	.85	4.09	10.25	11.75	12	48816	1.54	2.87	5.81	6.97
1000	101000	25065	12557	0	-	-	-	-	-	-	5	174509	5.67	7.49	17.33	19.72
1000	101000	50051	13699	0	-	-	-	-	-	-	1	407611	11.98	11.98	37.79	37.79
2500	27500	24957	3907	0	-	-	-	-	-	-	12	31363	2.20	3.00	2.76	3.11
2500	27500	62446	4705	0	-	-	-	-	-	-	12	90001	7.42	10.42	6.25	6.95
2500	27500	124902	5291	0	-	-	-	-	-	-	12	204891	14.35	19.12	13.05	14.93
2500	27500	249558	6033	0	-	-	-	-	-	-	12	462105	33.36	40.82	27.24	30.60
2500	65000	25060	6577	0	-	-	-	-	-	-	12	42551	3.30	4.30	4.06	4.70
2500	65000	62503	8944	0	-	-	-	-	-	-	12	143481	12.50	14.12	11.03	12.10
2500	65000	125196	10709	0	-	-	-	-	-	-	10	356271	25.85	31.79	24.98	27.86

Global execution time is made up of two parts : the execution time of the algorithm, and memory management time. Algorithm GPLA clearly dominates algorithm SPTW for the total time. This domination is mainly due to better memory management, and this will be true for all computers functioning with limited memory or virtual memory. This factor becomes particularly important for large problems.

The more detailed results of Table 2 show that for certain specific cases, algorithm SPTW is sometimes as efficient or more so than GPLA. The main factors leading to satisfactory performance of SPTW are :

1. a complete initialization of the list of nodes to be treated,
2. few efficient solutions per node (less than 5).

These two conditions ensure few reinsertions and render execution times for SPTW comparable to those for GPLA. If the two conditions are not satisfied, SPTW requires execution times which are, in several cases, twice those of GPLA. In a column generation context when the algorithm choses may be executed several hundred times, these performance differences may be very important.

8. CONCLUSION

A theorem on the order of treatment of labels, and the concept of a generalized bucket are used to develop an algorithm (GPLA) to solve the shortest path problem with time windows in pseudo-polynomial time. This new algorithm may also be used to solve shortest path problems with one additional constraint or with a double inequality. These two problems correspond to the case of wide time windows.

The algorithm GPLA solved efficiently all test problems in contrast to SPTW which performed badly on a fairly large number of problems as indicated in the analysis of the numerical results.

REFERENCES

[1] ANEJA, Y.P., AGGARWAL, V. and NAIR, K.P.K., Shortest chain subject to side constraints. Networks 13 (1983) 295-302

[2] BEASLEY, J.E. and CHRISTOFIDES, N., An algorithm for the resource constrained shortest path problems. Department of Management science, Imperial College, London (1986) 26 pages.

[3] DENARDO E.V., and FOX, B.L., Shortest-route methods : 1. reaching, pruning and buckets. Operations Research 27 (1979) 161-186.

[4] DESROSIERS, J., PELLETIER, P. and F., SOUMIS, Plus court chemin avec contraintes d'horaires. R.A.I.R.O. Recherche Opérationnelle 17 (1983) 357-377.

[5] DESROSIERS, J., SOUMIS, F. and DESROCHERS, M., Routing with time windows by column generation. Networks 14 (1984) 545-565.

[6] GALLO, G. and PALLOTTINO, S., Shortest Path Methods, in Transportation Models, M. Florian, Ed., Elsevier Science Press (1984) 227-256.

[7] GAREY, M.R. and JOHNSON, D.S., Computers and Intractability, A Guide to the Theory of NP-COmpleteness, Freeman, San Francisco (1979).

[8] HANDLER, G.Y. and ZANG, I., A dual algorithm for the constrained shortest path problem. Networks 10 (1980) 293-310.

[9] JAFFE, J.M., Algorithms for finding paths with multiple constraints. Networks 14 (1984) 95-116.

[10] JOKSCH, H.C., The shortest route problem with constraints. Journal of Mathematical Analysis and Applications 14 (1966) 191-197.

[11] MARTINS, E.Q.V., On a multicriteria shortest path problem. European Journal of Operational Research 16 (1984) 236-245.

[12] MINOUX, M., Plus court chemin avec contraintes : algorithmes et applications. Annales des Télécommunications 30, 12 (1975) 1-12.

[13] RIBEIRO, C.C.C., Algorithmes de recherche de plus courts chemins avec contraintes : étude théorique, implémentation et parallélisation. Thesis presented at École Nationale Supérieure des Télécommunications, Paris (1983) 237 pages.

[14] SORENSEN, B., Interactive Distribution Planning, Ph.D. Thesis, #46, IMSOR, Techinical University of Denmark (1986) 260 p.